

Desenvolvimento de **Serviços e APIs**

Validação de Dados com Zod

Curso Superior de Tecnologia em
Análise e Desenvolvimento de Sistemas
Prof. Edécio Fernando Iepsen



Sumário

- 1 Introdução ao Zod e sua importância
- 2 Instalando e configurando o Zod
- 3 Criando esquemas de validação
- 4 Validando dados no corpo da requisição (req.body)
- 5 Campos Opcionais e Enums
- 6 Valores default
- 7 Validando datas, strings e números
- 8 Exercícios

Introdução ao Zod

- O que é o Zod?
 - Biblioteca TypeScript-first para validação e tipagem de dados.
 - Permite definir esquemas de validação para garantir a integridade dos dados recebidos.
- Por que usar Zod?
 - Simples de usar.
 - Integra-se bem com APIs Express.
 - Ajuda a evitar erros de dados inesperados.

Instalação e Uso

1

Instalação

```
npm install zod
```

2

Uso

```
import { z } from "zod"
```



Zod

zod.dev

TypeScript-first schema validation with static type inference

Criando Esquemas de Validação

Esquema Básico:

```
const usuarioSchema = z.object({  
  nome: z.string().min(3),  
  email: z.string().email(),  
  idade: z.number().min(18)  
});
```



Esse esquema valida que:

- nome é uma string de no mínimo 3 caracteres.
- email é um endereço de e-mail válido.
- idade é um número maior ou igual a 18.

Validando req.body dentro da Função da Rota

```
app.post('/usuarios', (req, res) => {  
  const result = usuarioSchema.safeParse(req.body);  
  if (!result.success) {  
    res.status(400).json(result.error.format());  
    return  
  }  
  
  // Desestrutura os dados validados  
  const { nome, email, idade } = result.data;  
});
```

Campos Opcionais:

```
const usuarioSchema = z.object({  
  nome: z.string().min(3),  
  ...  
  // Pode ser string, null ou ausente  
  cidade: z.string().nullable().optional()  
})
```

```
// Na função de inclusão
```

```
...  
const { nome, cidade = null } = result.data
```

Campos Enum:

```
model Carro = {  
  ...  
  combustivel Combustiveis  
}
```

```
enum Combustiveis {  
  FLEX  
  GASOLINA  
  ...  
}
```

```
import { Combustiveis } from '@prisma/client'  
  
const carroSchema = z.object({  
  ...  
  combustivel: z.nativeEnum(Combustiveis)  
})
```


Valores default

```
model Carro = {  
  ...  
  combustivel Combustiveis @default(FLEX)  
}
```

```
enum Combustiveis {  
  FLEX  
  GASOLINA  
  ...  
}
```

```
const carroSchema = z.object({  
  ...  
  combustivel: z.nativeEnum(Combustiveis).optional()  
})  
  
// Na função de inclusão  
...  
const { ..., combustivel = "FLEX" } = result.data
```

Validando Datas:

```
const usuarioSchema = z.object({  
  nome: z.string().min(3),  
  ...  
  datanasc: z.string().date()  
})
```

Validação de Strings:

```
// validations
z.string().max(5);
z.string().min(5);
z.string().length(5);
z.string().email();
z.string().url();
z.string().emoji();
z.string().uuid();
z.string().nanoid();
z.string().cuid();
z.string().cuid2();
z.string().ulid();
z.string().regex(regex);
z.string().includes(string);
z.string().startsWith(string);
z.string().endsWith(string);
z.string().datetime(); // ISO 8601; by default only `Z` timezone allowed
z.string().ip(); // defaults to allow both IPv4 and IPv6
z.string().cidr(); // defaults to allow both IPv4 and IPv6

// transforms
z.string().trim(); // trim whitespace
z.string().toLowerCase(); // toLowerCase
z.string().toUpperCase(); // toUpperCase
```

Mensagens

```
z.string().min(5, { message: "Must be 5 or more characters long" });
z.string().max(5, { message: "Must be 5 or fewer characters long" });
z.string().length(5, { message: "Must be exactly 5 characters long" });
z.string().email({ message: "Invalid email address" });
z.string().url({ message: "Invalid url" });
z.string().emoji({ message: "Contains non-emoji characters" });
z.string().uuid({ message: "Invalid UUID" });
z.string().includes("tuna", { message: "Must include tuna" });
z.string().startsWith("https://", { message: "Must provide secure URL" });
z.string().endsWith(".com", { message: "Only .com domains allowed" });
z.string().datetime({ message: "Invalid datetime string! Must be UTC." });
z.string().date({ message: "Invalid date string!" });
z.string().time({ message: "Invalid time string!" });
z.string().ip({ message: "Invalid IP address" });
z.string().cidr({ message: "Invalid CIDR" });
```

Validação de Números:

Numbers

You can customize certain error messages when creating a number schema.

```
const age = z.number({  
  required_error: "Age is required",  
  invalid_type_error: "Age must be a number",  
});
```

Zod includes a handful of number-specific validations.

```
z.number().gt(5);  
z.number().gte(5); // alias .min(5)  
z.number().lt(5);  
z.number().lte(5); // alias .max(5)  
  
z.number().int(); // value must be an integer  
  
z.number().positive(); // > 0  
z.number().nonnegative(); // >= 0  
z.number().negative(); // < 0  
z.number().nonpositive(); // <= 0  
  
z.number().multipleOf(5); // Evenly divisible by 5. Alias .step(5)
```

Exercícios

- 1 Acrescentar um schema de validação na model Viagem
 - 2 Validar “estrelas” para que, além de número, fique no intervalo 1.. 5
 - 3 Validar “destino” para que aceite apenas os enums indicados
 - 4 Acrescentar mensagens nas validações
- 